8Ц

УДК 004.451.46

И.В. Царёв
Санкт-Петербургский институт информатики и автоматизации РАН,
г. Санкт-Петербург, Россия
civ@mail.iias.spb.su

ЯРД – язык сетевого программирования в распределенных вычислительных системах с динамической архитектурой

Рассматривается сетевой язык программирования, предназначенный для параллельного программирования в распределённых системах, основанных на идеологии мультипроцессоров с динамической архитектурой (МДА), обеспечивающих автоматическое распараллеливание программ, высокую надежность вычислений и неограниченную масштабируемость. Рассматриваемый язык описывает не алгоритм решения задачи, а её структуру, которая изменяется в процессе решения задачи. Язык имеет две формы представления задачи – графическую и текстовую и полностью освобождает программиста от необходимости заботиться о распараллеливании задачи и об используемых вычислительных ресурсах.

Введение

Целью данной статьи является автоматизация программирования и выполнения параллельных программ в среде мультипроцессоров с динамической архитектурой (МДА), которые ориентированы не на конкретные аппаратные решения, но на структуру выполняемой задачи.

Основной задачей является организация и разработка таких программных и аппаратных средств, которые в большей степени соответствовали бы не столько традиционной архитектуре аппаратных средств суперкомпьютеров, сколько структуре решаемой задачи, что может способствовать существенному повышению эффективности суперкомпьютеров. Важнейшим фактором такого повышения эффективности параллельных вычислений является решение проблемы «семантического разрыва». В данной статье рассматриваются не аппаратные, а программные средства решения этих проблем, хотя в рамках МДА они весьма тесно взаимосвязаны.

Решение проблемы «семантического разрыва»

Существует широко известное понятие «семантического разрыва» (СР), введённое Г. Майерсом [1], которое охватывает многие аспекты решения различных задач на различных вычислительных средствах. Чаще всего это понятие ассоциируется с разработкой компиляторов. В этом случае рассматривается СР между существующими языками программирования высокого уровня и архитектурой ЭВМ, основанной на традиционных и общеизвестных идеях фон-Неймана [2]. Собственно говоря, в этом случае СР заключается в концептуальных различиях между программными структурами языков высокого уровня и архитектурой и системой команд конкретной ЭВМ, для которой разрабатывается компилятор с некоторого языка программирования.

Попытки преодолеть этот аспект СР предпринимались неоднократно, включая как отечественные разработки (например, МВК «Эльбрус»), так и зарубежные (некоторые разработки фирмы «Burroughs»), которые поддерживали на уровне архитектуры аппарат-

ных средств основные структуры языков программирования высокого уровня. Однако это не привело к кардинальному решению данной проблемы, поэтому эти проекты так и не получили большого распространения.

На наш взгляд, основная причина этого заключается в другом уровне СР, а именно: СР между алгоритмической (т.е. изначально последовательной) моделью вычислений (для которой относительно приспособлено большинство существующих компьютеров), на которой также основано большинство существующих языков программирования, и реальной структурой решаемой задачи, которая далеко не всегда является линейной и последовательной.

Эта проблема ещё больше усугубляется в современных многопроцессорных суперкомпьютерах, поскольку алгоритмическая модель совершенно не приспособлена для решения многих реальных задач, многие из которых имеют изначальную способность к распараллеливанию вычислительного процесса.

Всё это является причиной низкой эффективности большинства современных суперкомпьютеров и распределённых вычислительных систем, поскольку все они ориентированы на решение задачи посредством искусственного распараллеливания процессов (при наличии соответствующих программных средств). Эти средства позволяют относительно неплохо распараллеливать задачи, решение которых связано с регулярными векторными и матричными структурами, системами алгебраических или дифференциальных уравнений, однако даже на таких задачах (например, общеизвестная тестовая задача «Linpack», на основе результатов решения которой формируется список 500 наиболее производительных суперкомпьютеров – Top-500), как правило, суперкомпьютеры из этого списка показывают эффективность вычислительного процесса на тесте «Linpack» в среднем порядка 70 – 80 % от пиковой (т.е. теоретической) производительности. Что же касается более сложных реальных задач, то на них реальная производительность падает на 1 – 2 порядка. К последним относятся и многие задачи искусственного интеллекта.

Кроме того, эти средства возлагают на программиста множество проблем, связанных с распараллеливанием программ, изначально основанных на последовательных алгоритмах, синхронизацией процессов и т.д. Это требует очень высокой квалификации программистов и значительных усилий по программированию параллельных процессов, которые могут быть сведены «к нулю» при изменении параметров или конфигурации параллельной вычислительной системы.

Предлагаемая система мультипроцессоров с динамической архитектурой (МДА), основанная на ДАС, а также рассматриваемый ниже язык ЯРД позволяют решить эту проблему (семантического разрыва), поскольку архитектура МДА «подстраивается» под структуру задачи, а не под существующие в ней аппаратные средства.

Динамические автоматные сети

В 80-х годах профессором В.А. Торгашёвым была предложена новая модель вычислений, основанная на динамических автоматных сетях (ДАС) [3], которая впоследствии позволила реализовать ряд образцов мультипроцессоров с динамической архитектурой (МДА).

Любая программа в МДА – это ДАС, элементами которой являются автоматы, реализующие функции семи основных классов программных структур (автоматов): **операторы, данные, отношения, ссылки, ресурсы, типы** и **структуры (подсети)**. Каждый автомат обладает способностью изменять свои связи с окружающими его автоматами и порождать новые автоматы, а также самоуничтожаться после выполнения своих функций.

Вычисление в МДА сводится к автотрансформации автоматной сети (ДАС), которая в начальной стадии вычислений растёт за счёт создания (генерации) новых автоматов (узлов программной сети), выполняемых параллельно на различных ресурсах (например, процессорах) вычислительной системы, а затем, по мере решения задачи, уменьшается за счёт уничтожения автоматов, выполнивших свою функцию, причем результатом вычисления является сеть, потерявшая способность к изменениям, либо в качестве результата может рассматриваться воздействие изменяющейся автоматной сети на внешнюю среду. При этом в МДА осуществляется полная децентрализация управления как на аппаратном, так и на программном уровнях. В МДА реализуется динамическое автоматическое распараллеливание вычислений и распределение ресурсов при отсутствии принципиальных ограничений на количество и число типов этих ресурсов (процессоров, внешних устройств, связей, оперативных запоминающих устройств и т.д.). В результате архитектура МДА динамически подстраивается под структуру решаемой задачи, что позволяет в значительной степени решить проблему СР.

Язык программирования «ЯРД»

На уровне программирования автором данной идеи (ДАС/МДА) В.А. Торгашёвым был в 80-х годах разработан язык РЯД [4]. Однако этот язык имел не слишком удачный синтаксис и не полностью определённую семантику, поэтому не был успешно реализован.

На основе этих идей автором данного сообщения в середине 90-х годов был разработан и реализован в виде транслятора новый неалгоритмический, сетевой язык ЯРД (аббревиатура расшифровывается как Язык Рекурсивный Динамический, впрочем, аббревиатура «РЯД» расшифровывается точно так же, только с перестановкой слов). Впервые язык «ЯРД» (первый его вариант) был представлен на международной конференции РАСТ-93 [5]. Язык «ЯРД» использовался для программирования различных задач на макетных образцах МДА, которые в середине 90-х годов прошлого века были реализованы на основе процессоров серии TMS320 и появившихся в то время простейших схем гибкой логики (PLD). Более поздние разработки МДА основаны на более современных ПЛИС фирмы «Altera» и без использования серийных процессоров, что позволяет реализовать одну из основных идей МДА – каждый узел программной сети (программный аналог узлов – автоматов ДАС) фактически является виртуальным процессором, который создаётся только на время выполнения своей функции, а после выполнения оной – уничтожается, что позволяет в значительной мере экономить аппаратные ресурсы.

Со времени создания первых версий языка ЯРД [5], [6] он был существенно доработан (в части текстовой формы представления, семантики и реализации). В рамках краткого сообщения невозможно, да и не имеет смысла полностью рассматривать синтаксис и семантику языка, поэтому остановимся на основных особенностях языка ЯРД и его отличиях от традиционных языков программирования.

Прежде всего этот язык – сетевой, отражающий не алгоритм решения задачи, а её структуру – сеть, состоящую из объектов (чаще всего данных и операторов), связанных различными отношениями, среди которых наиболее важными и наиболее часто встречающимися являются отношения «аргумент-результат». В таком виде естественным образом может быть представлено большинство задач, решаемых на мультипроцессорных компьютерах. В процессе выполнения программы структура и вид сети изменяются, так что программа описывает только начальную структуру задачи, а также может описывать некоторые аспекты преобразования программной сети и включённых в сеть объектов. Последнее необходимо только в тех случаях, когда поведение объекта в автотрансформирующейся сети отличается от стандартного для данного класса объектов.

Язык имеет две взаимосвязанных и взаимозаменяемых формы представления программы – графическую и текстовую. В графической форме задача представляется в виде графа, каждому узлу которого соответствует виртуальный процессор (а в задаче – некоторый объект), а связи между ними – отношениям между узлами (объектами). Каждый узел программной сети относится к одному из семи классов, о которых говорилось выше, при этом каждый класс имеет своё собственное графическое изображение. Поскольку отобразить все нюансы и свойства конкретного узла только в графической форме практически невозможно, то при формировании графического изображения программной сети автоматически генерируется и текстовая форма, в которую можно вносить изменения и дополнения, но можно и полностью писать программу в текстовой форме, при этом автоматически генерируется и её графическое представление.

Каждый объект и сеть в целом имеют однозначную графическую интерпретацию в графической версии языка, которому соответствует адекватное текстовое представление. Хотя данный текст не является описанием графической версии языка, тем не менее в тексте приведены рисунки, иллюстрирующие графическую форму объектов и сетевых программ [7]. На рис. 1 приведены графические обозначения семи базовых классов объектов. Каждому классу объектов соответствует геометрическая фигура. Внутри фигуры может быть вписано текстовое обозначение конкретного объекта (например идентификатор или обозначение операции), имя его конкретного типа, либо пиктограмма (иконка), введенная в графической программе для обозначения этого объекта или его типа. Обозначения объектов соединяются линиями (дугами), соответствующими связям объекта. Если связь односторонняя, то она может быть снабжена стрелкой, указывающей направление связи (стрелка указывает на объект, для которого связь является входной). Справа от наименований классов узлов, через тире, приведены соответствующие обозначения в текстовой форме программы (ключевые слова).

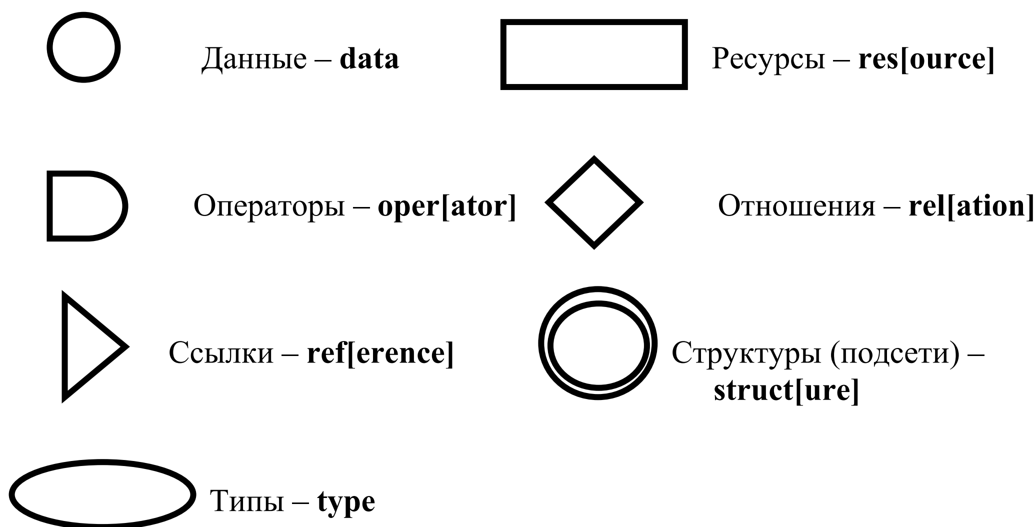


Рисунок 1 – Обозначения классов объектов в графической версии языка ЯРД

В текстовой версии языка объект, снабженный связями, может обозначаться идентификатором, за которым следует список объектов, связанных с данным, заключенный в скобки (список связей). Объекты, связанные с данным (их можно называть, например, аргументами и результатами, а в более общем случае – фактическими параметрами) в списке следуют в том порядке, в котором следуют соответствующие связи в формальном описании объекта. В целом это имеет вид, напоминающий вызов функции в традиционных языках программирования. Существенное отличие от традиционных языков заключается в том, что основной идентификатор в такой записи может при-

надлежать объекту любого класса, в том числе и класса «данные», а аргументом такого объекта может быть и оператор. Связь объекта с объектом-типом, определяющим свойства и поведение данного объекта, в текстовой форме изображается двоеточием (‘**:**’), так что это становится похоже на обозначение типа в языке «Паскаль» или ключевым словом **is** (например, X : matrix или Z **is** matrix), а связь с ресурсом (модули/процессоры распределённой системы, периферийные устройства, специально зарезервированные блоки памяти и т.п.) изображается ключевым словом a**t** или символом ‘**@**’, например, X a**t** input или Z **@** output.

Программа в целом является некоторой специфической разновидностью структуры. Поэтому программа обозначается ключевым словом **prog[ram]**, семантика которого является скрытой (имеет значение только для операционной системы и аппаратных средств), в синтаксисе все остальные части программы не отличаются от соответствующих частей структуры (подсети). Программу легко включить в состав другой, более крупной программы, всего лишь заменив ключевое слово в её описании (**prog** на **struct**) и перенаправив входные и выходные связи от периферийных устройств к другим объектам внешней программы, вырабатывающим или потребляющим соответствующие объекты (аргументы/результаты).

На рис. 2 изображен фрагмент программной сети, содержащий четыре объекта: оператор **mult** (например, это может быть оператор умножения матриц или векторов), два его аргумента A и B и результат C (о семантике этих объектов мы пока не говорим, для простоты пока рассматриваем только данные и операторы в обобщённой форме, но по-видимому, это матрицы или векторы). В текстовой форме языка этот фрагмент сети может быть записан следующим образом:

mult (A, B, C); A (X, mult); B (Y, mult); C (mult, Z);

где идентификаторами X, Y и Z обозначены объекты, не показанные на рисунке (источники аргументов и приемники результатов – это могут быть как узлы класса «ресурс», например входные и выходные устройства, так и другие операторы программы, которые эти значения вырабатывают или потребляют), внешние по отношению к данному фрагменту сети.

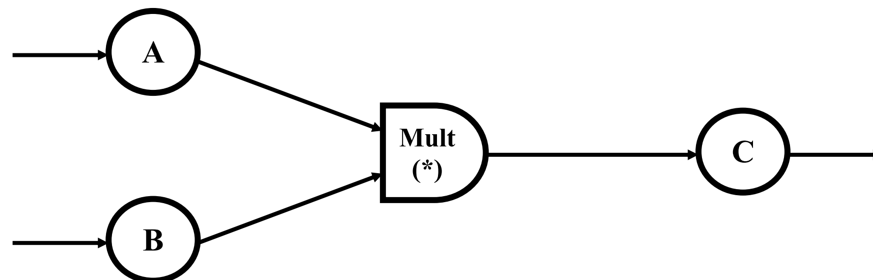


Рисунок 2 – Фрагмент вычислительной сети

Приведенный фрагмент сети может быть заключен в скобки и в этом случае он становится структурным узлом (структурой), содержащей в себе подсеть из четырех объектов:

```
(mult (A, B, C); A (X, mult); B (Y, mult); C (mult, Z);)
   или даже
begin
   mult (A, B, C); A (X, mult); B (Y, mult); C (mult, Z);
end;
```

Хотя простые скобки и операторные скобки **begin** ... **end** в большинстве случаев семантически равнозначны, все же в реализации они могут различаться. Различие состоит в том, что подсеть, заключенная в скобки **begin** ... **end** всегда оформляется (на внутреннем языке системы) как отдельный структурный узел, «внутри» которого заключена подсеть, при этом внешние связи подсети «подключены» к связям структурного узла как «транзитные» связи (термин из внутреннего языка системы, обозначающий связи с внешними объектами, которые не включены в данный фрагмент программы).

Однако язык ЯРД позволяет записать этот фрагмент программы и гораздо более кратко, в виде выражения, например, так:

C **at** Z := A **at** X **mult** B **at** Y; или

C **@** Z := A **@** X * B **@** Y;

Приоритет связей с ресурсом или типом всегда выше, чем приоритет всех остальных операций и операторов, а второй выше первого. Приоритеты обычных арифметических и логических операций составляют вполне стандартную иерархию. Впрочем, любые приоритеты операций и обозначений в языке могут быть изменены при помощи описания приоритета (ключевое слово **prio[rity]**).

Более того, связи аргументов и результатов с ресурсами и типами могут быть показаны и в описаниях объектов, например, так:

C **:** matrix **@** Z; A **is** matrix **at** X; B **is** matrix **at** Y;

Тогда само выражение будет записано ещё более кратко:

C := A * B;

Обозначение, аналогичное оператору присваивания '**:=**', здесь означает лишь связь основного оператора выражения ('*') с получателем вычисленного значения (результатом) С. Эта связь всегда имеет наинизший приоритет из всех возможных.

Язык также содержит и некоторые конструкции, подобные обычным языкам программирования, таким как «условная конструкция» (**"if-then-else"**), «конструкция выбора» («переключатель») – "**case**" или "**switch**" и т.п. Семантика этих конструкций также отличается от традиционной. Реально генерируются все возможные узлы ветвей, в соответствии с описываемыми программными структурами (т.е. порождается сеть, состоящая из «ветвей», каждая из которых связана отношением «аргумент» с узлом, представляющим условие или аргумент выбора), но только одна из них выполняется, в зависимости от значения условия (или аргумента выбора/переключателя), после момента, в который единственный соответствующий аргументу выбора (условия) узел получит «истинное» состояние, остальные приобретают «ложное» состояние, которое приводит к уничтожению этих узлов или подсетей (которые так и не выполняются, но самоуничтожаются). Таким образом, условные конструкции и конструкции выбора порождают сети, в которых только одна из ветвей выполняется, а остальные уничтожаются (независимо от внутренней сложности ветвей).

Несколько сложнее с таким понятием, как «циклы». В отличие от традиционных языков программирования, «ЯРД» не имеет такой конструкции, как «цикл», поскольку язык не алгоритмический, а сетевой, хотя имеется синтаксическая конструкция, начинающаяся на ключевое слово "**for**", которое традиционно, в обычных языках, и обозначает цикл.

Для управления множественными вычислениями используются так называемые «кванторы», в текущей версии языка их всего два – '**all**' («все») и '**any**' («любой»). Кванторы применяются как к элементам сложных структур или массивов (включая многомерные, в этом случае квантор может заменять один или несколько индексов массива

или селекторов структуры), так и к элементам программной сети, означая некоторый выбор из частей массивов или структур, программных ветвей, частей параллельного выполнения программы, ресурсов. Первый означает применение некоторой конструкции языка ко всем элементам некоторой подчинённой структуры (включая ресурсы), второй – к одной, но любой из потенциально возможных структур. Например, в отношении целой программы или замкнутой её части («структуры», «узла», «процедуры») сочетание символов "**at all**" означает размещение всей программы или её части на всех существующих аппаратных ресурсах системы (модулях, процессорах), а "**at any**" – на одном (произвольном) из всех доступных ресурсов. По отношению к элементам массива или сложной структуры квантор может означать выбор либо всех элементов структуры (в этом случае вычисления автоматически распараллеливаются в соответствии с количеством этих элементов, а подструктуры программы автоматически размножаются). Либо только одного из них, произвольного. В последнем случае выполняется только один из элементов, произвольный.

Если рассматривать обычные циклы в традиционных языках программирования, то их однозначно можно разделить на два основных типа – циклы с перечислением (обычно это циклы, изображаемые при помощи ключевого слова '**for**'), в которых содержание вычислений в цикле зависит только от значений переменной цикла, которая чаще всего обозначается как 'i', во вложенных циклах – как 'j', 'k' и т.д. (хотя принципиально возможны и любые другие обозначения переменной цикла), а также итеративные циклы (циклы типа '**while**', '**repeat**'), в которых значение переменной цикла может зависеть от вычислений на предыдущем цикле.

Аналог первого случая в языке «ЯРД» состоит в генерации множества подсетей, соответствующих содержимому **for**-конструкции, которые будут автоматически распределены по всем возможным аппаратным ресурсам и по мере возможности будут выполняться параллельно. Например,

for all i **do** C [i] := …

Конкретно, в данном случае, для всех возможных значений индексной переменной 'i' будут сгенерированы соответствующие подсети, которые, вычислив параллельно соответствующие значения на различных ресурсах системы (модулях, процессорах), присвоят полученные вычисленные значения соответствующим элементам массива C.

Второй случай (итеративный цикл) более сложен, поскольку в итеративных циклах обычно имеется некоторая зависимость между вычислениями в последовательных ветвях (например, X [i+1] := f (X [i])). В этом случае в МДА будет сгенерировано множество подсетей, которые будут связаны (прямо или косвенно) отношением следования, т.е. следующая подсеть не может быть выполнена, пока не выполнится предыдущая (или в ней не будет выработан очередной параметр). Операционная система МДА не позволяет генерировать одновременно такое множество ветвей программной сети, которое перегрузило бы ресурсы системы, так что ветви будут генерироваться по мере освобождения ресурсов уже выполненными фрагментами программы.

В любом случае, **for**-конструкция в языке «ЯРД» означает не цикл, а параллельное выполнение ветвей потенциального цикла, насколько это позволяют ресурсы конкретной системы.

Некоторые дополнительные свойства языка «ЯРД»

Поскольку свойства каждого объекта программной сети содержат и алгоритмы поведения этого объекта (включая и вычислительные), то язык является в полной мере объектно-ориентированным.

Текстовая форма программы похожа на традиционные языки программирования (изначально в большей степени это стиль языка «Паскаль», но транслятор позволяет перенастроить синтаксис таким образом, чтобы он соответствовал наиболее (хотя и не вполне точно) привычному для конкретного программиста языку, например, «Алгол-68» или «С++»). Но в любом случае, в отличие от традиционных языков программирования, порядок написания выражений безразличен, так как их выполнение не алгоритмическое, т.е. последовательное, а сетевое и параллельное.

Для этой цели (изменение стиля программирования) служит, с одной стороны, включённый в язык механизм описания синонимов, который позволяет заменять обозначения, например, заменить идентификатор оператора умножения матриц или векторов **mult** на более простое обозначение '*' (в этом случае действует механизм перекрытия функций/процедур/объектов с одинаковыми обозначениями, но с разными списками связей-параметров и разными типами аргументов), либо изменять написание любых слов (идентификаторов, ключевых слов) на любое другое (разумеется, если это не вносит противоречий). Это, в частности, позволяет и формировать национальные версии языка, не меняя транслятора. Например, описать русский синоним для идентификатора оператора **mult**:

syn[onym] умнож to mult;

Или описать функциональное обозначение для некоторого оператора (в данном примере – инкремента) в виде обозначения операции (предполагается, что оператор **inc** описан в виде процедуры/функции (либо встроен в систему), причём возможно наличие нескольких описаний этой функции, например, как инфиксной, постфиксной и префиксной, синоним только заменяет обозначение):

syn[onim] ++ to inc;

Кстати, и приведённые в некоторых примерах вариации обозначений идентификаторов, ключевых слов (необязательные части в квадратных скобках), например **syn** вместо **synonym** или **struct** вместо **structure** также определены как синонимы, но уже заранее встроенные в язык.

С другой стороны, для этой цели может использоваться пока ещё не вполне доработанный и пока не полностью реализованный механизм, который позволяет непосредственно в программе определять (изменять) синтаксис некоторых программных структур (в пределах некоторого синтаксического блока).

Однако следует учитывать, что семантика языка, несмотря на внешнее сходство синтаксиса, принципиально отличается от традиционных языков программирования, поскольку представляет собой лишь описание начальной программной сети, соответствующее структуре задачи, а также средства и способы изменения этой структуры в процессе решения.

Описание каждого объекта в программе может содержать различные секции, которые определяют различные аспекты поведения объекта в соответствии с идеологией МДА – вычисление, управление (принятие решений о функционировании объекта на разных стадиях решения задачи) и коммутацию (порождение новых объектов, уничтожение объектов, выполнивших свою функцию, изменение связей, пересылка объектов в другие аппаратные ресурсы и т.д.). Впрочем, определение управления и коммутации для большинства объектов не является необходимым, так как в большинстве случаев достаточно стандартных средств, присущих конкретному классу объекта, которые естественным образом наследуются.

Описание функций коммутации и управления возможно только в текстовой форме программы.

Для программирования параллельных (распределённых) вычислений в МДА не требуется знать, сколько процессоров входит в состав машины и какова структура связей между процессорами. Программист полностью освобождён от привязки к конкретной архитектуре распределённой сети компьютеров или мультипроцессорной машины, а также от необходимости предпринимать какие-то специальные действия по распараллеливанию программы, это происходит автоматически.

В то же время язык (и аппаратно-программная реализация системы) позволяет при необходимости как явным образом осуществлять распределение данных или частей программы между имеющимися в наличии ресурсами системы (при использовании «связей с ресурсами»), так и осуществлять при необходимости явную синхронизацию процессов или привязку их к конкретным моментам времени или к событиям, происходящим в программе или во всей системе. Последнее позволяет реализовывать на данной основе и системы реального времени, например, системы обработки сигналов (подобные программы на существовавших в то время реализациях МДА для обработки в реальном времени гидроакустических сигналов и телеметрической информации со спутников и космических кораблей были реализованы в 90-х годах).

Свойства конкретных классов объектов языка «ЯРД» в значительной степени определяют и функционирование операционной системы, часть функций которой реализуется аппаратно, а часть является свойствами самих классов объектов, заложенных как в аппаратуру и операционную систему, так и в реализацию некоторых конструкций языка (последнее – в виде включённых в описание объекта процедур-методов).

Подобная структура языка и соответствующих аппаратных средств МДА позволяет эффективно решать практически любые задачи, включая и сложные по структуре задачи искусственного интеллекта.

Заключение

При использовании языка «ЯРД» в аппаратно-программной среде МДА решается сразу несколько задач:

– относительно простой и естественный способ представления программы (задачи) в сетевой (графической) форме, соединённый с возможностью текстового описания конкретных свойств элементов программной сети (объектов);

– возможность представления задачи (программы) как в графической и текстовой, так и исключительно в текстовой форме, в зависимости от предпочтений конкретного разработчика;

– возможность лексической (а в последствии и синтаксической) настройки языка в зависимости от предпочтений программиста;

– многовариантность представления задачи в текстовой форме;

– автоматическое распараллеливание задачи в соответствии с её структурой, а не структурой аппаратных средств;

– практически полное освобождение программиста от необходимости заботиться об явном распараллеливании вычислительного процесса, от привязки процессов к моментам времени или событиям и о синхронизации процессов, если это явно не требуется в конкретной задаче (например, в задачах реального времени), но для этого в языке имеются соответствующие средства;

– практически полное освобождение программиста от необходимости знать в момент разработки программы о количестве ресурсов (модулей, процессоров, памяти) в системе и связях между ними (т.е. об аппаратной структуре системы);

– возможность определять при необходимости поведение объектов программы (функции управления и коммутации);

– максимально эффективное (в зависимости от задачи) использование аппаратных ресурсов системы;
– возможность программирования параллельных задач не только в рамках специально разработанного суперкомпьютера, основанного на идеологии МДА, но и использования ресурсов любой ПЭВМ, включённой в состав grid-системы или метакомпьютера для организации распределённых вычислений;
– обеспечение максимальной информационной безопасности распределённой вычислительной системы [7].

Таким образом, использование программно-аппаратного комплекса МДА, в основе программирования которого находится вышеописанный язык «ЯРД», позволяет решить большинство основных проблем использования многопроцессорных аппаратных средств и параллельного программирования для множества нерегулярных задач, чего не делает большинство существующих многопроцессорных систем и систем параллельного программирования.

Литература

1. Майерс Г. Архитектура современных ЭВМ: В 2 кн. – М.: Мир, 1985. – Кн. 1. – 364 с.
2. J. von Neuman. Theory of self-reproducing automata. – University of Illinois Press, Urbana and London, 1966.
3. Торгашев В.А., Царев И.В. Средства организации параллельных вычислений и программирования в мультипроцессорах с динамической архитектурой // Программирование. – 2001. – № 4. – С. 53-68.
4. Торгашев В.А. РЯД – Язык программирования для распределенных вычислений. Препринт № 28 / Академия наук СССР. Ленинградский научно-исследовательский вычислительный центр. – Ленинград. – 1984. – С. 3-48.
5. Torgashev V.A. and Tsaryov I.V. A Parallel Programming Language for Dynamic Architecture Computers // In Proc. of The Int. Conf. Parallel Computing Technologies. ReSCo. – Moscow. – 1993. – P. 31-34.
6. Царев И.В. Языковые средства и функции операционной системы в мультипроцессорах с динамической архитектурой // Искусственный интеллект. – 2003. – № 4. – С. 66-73.
7. Царев И.В. Аппаратно-программные методы обеспечения надежности вычислений в мультипроцессорах с динамической архитектурой // Известия ТРТУ. Специальный выпуск. – Таганрог: Изд-во ТРТУ. – 2005. – С. 61-70.

І.В. Царьов
ЯРД – мова сітьового програмування у розподілених обчислювальних системах з динамічною архітектурою
Розглядається сітьова мова програмування, призначена для паралельного програмування в розподілених системах, заснованих на ідеології мультипроцесорів з динамічною архітектурою (МДА), які забезпечують автоматичне розпаралелювання програм, високу надійність обчислень і необмежену масштабованість. Дана мова описує не алгоритм рішення задачі, а її структуру, яка змінюється в процесі рішення задачі. Мова має дві форми представлення задачі – графічну і текстову і повністю звільняє програміста від необхідності піклуватися про розпаралелювання задачі і про використовувані обчислювальні ресурси.

I.V. Tsaryov
YARD – Network Programming Language for Distributed Computing Systems with Dynamic Architecture
Network programming language, dedicated for parallel programming in distributed systems based on ideology of multiprocessors with dynamic architecture (MDA) that provide automatic paralleling programs, high reliability of computations and unlimited scaling is considered. The language considered describes not the algorithm of the problem being solved but its initial structure that can be transformed during the process of solution. The language has two forms of program representation – graphic and text ones and completely releases the programmer from care of paralleling program and of computational resources used.

Статья поступила в редакцию 21.07.2008.